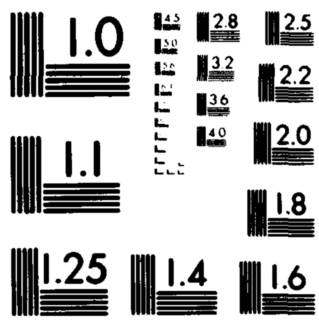
AD-A083 266 VIRGINIA POLYTECHNIC INST AND STATE UNIV BLACKSBURG --ETC F/G 9/2
CONCERNING CLASSES WITHIN CLASSES.(U)
JAN 79 R J ORGASS AFOSR-79-0021
UNCLASSIFIED VPI-TM-79-1 AFOSR-TR-80-0282 NL

1 OF 1
AD A
083 266

END
DATE FILMED
5-80
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AFOSR-TR- 80-0282

(12) EXTENSION DIVISION

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

LEVEL II

ADA083266

CONCERNING CLASSES WITHIN CLASSES

Richard J. Orgass

Technical Memorandum No. 79-1

January 15, 1979

AFOSR-79-0021

ABSTRACT

Two examples are used to show how the SIMULA restriction on the use of dot notation to reference attributes of classes limits the abstraction mechanism provided by classes.

Key Words: SIMULA, program abstraction

CR Categories: 4.22, 5.24

Located at Dulles International Airport—400 West Service Road

CONCERNING CLASSES WITHIN CLASSES

The SIMULA Common Base Definition [1], Section 7.1.2, restricts the use of dot notation to refer to attributes of classes as follows:

> *The remote identifier X.A is valid if the following conditions are satisfied:*
>
> *1) The value X is different from none.*
>
> *2) The object referenced by X has no class attribute declared at any prefix level equal or outer to that of C.*

While the restriction (2) makes it a great deal easier to implement SIMULA, it also serves to make it much more difficult to use classes as an abstraction mechanism and to share concepts implemented as classes.

I'd like to illustrate the difficulty with two examples. The first is from a program that I have been working with for some time. In this program, there is a need for linear lists of unknown length. Different lists have different kinds of nodes and different attributes are of interest. All of these lists share some attributes which are not accessible outside the structures. Thus, it is reasonable to declare:

```
class linked_list;

protected l_l_node,
          insert,
          head;

begin

    class l_l_node;
    begin
        ref(l_l_node) link
    end of l_l_node;

    procedure insert (element); ref(l_l_node) element;
    begin
        element.link :- head;
        head :- element
    end of insert;

    ref(l_l_node) head;

end of linked_list;
```

This class declaration is used for many structures which include stacks of tokens and stacks of reals. The class token_stack might be declared as follows:

```
linked_list class token_stack;

not protected push,
              pop;

begin

    l_l_node class exp_node(v); ref(token) v;
        begin end;

    procedure push(id); ref(token) id;
        insert (new exp_node(id));

    ref(token) procedure pop;
        if head =/= none
           then begin
                     pop :- head qua exp_node.v;
                     head :- head.link
                end
           else pop :- none;
end of token_stack;
```

This declaration provides stacks of tokens and a pop executed on an empty stack returns the object none. This return value is tested by the program that uses the stack.

A similar declaration of class real_stack is:

```
linked_list class real_stack;

not protected push,
              pop;

begin

    l_l_node class real_node(v); real v;
        begin end;

    procedure push(val); real val;
        insert(new real_node(val)):

    real procedure pop;
        if head =/= none
           then begin
                    pop := head qua real_node.v;
                    head :- head.link
                end
           else ("popping empty real stack.");
end of real_stack;
```

The nodes placed on this stack differ from the nodes on the token stacks. In addition, popping an empty stack of reals is a fatal run time error and execution is terminated in this case.

These declarations make it possible to state the difficulty clearly. It is possible to use the push and pop attributes of either a token_stack or a real_stack as follows:

```
inspect new token_stack do
    begin ... end;
```

or

```
inspect new real_stack do
    begin ... end;
```

However, it is not possible to use both stacks simultaneously because the following is illegal by restriction (2).

```
ref(token) y;
real z;
ref(token_stack) t_s;
ref(real_stack) r_s;
⋮
if r_s.pop < 0
   then t_s.push(y)
   else r_s.push(z);
```

There are many obvious technical devices for avoiding this difficulty but they all make it difficult to define objects in terms of other objects. For example, one could rearrange the declarations and duplicate code or replace

```
t_s.push(y)
```

with

```
inspect t_s do push(y)   .
```

Both of these solutions as well as others make it difficult to use classes as an abstraction mechanism and they are forced by restriction (2).

As a second example, which is typical of data structure definitions, arises as follows. Suppose a stack of integers is to be implemented with four procedure attributes push, pop, full and empty. The class declaration, as it concerns the user of these stacks is:

```
class stack(n); value n; integer n
not protected push,
              pop,
              full,
              empty;

begin
    ⋮
    procedure push(x); value x; integer x;
    ⋮
    integer procedure pop;
    ⋮
    boolean procedure full;
    ⋮
    boolean procedure empty;
    ⋮
end of stack;
```

In some cases, it may be desirable to implement this kind of stack with an array and in other cases it may be desirable to implement this kind of stack using linked lists.

If the stacks are implemented as arrays, the procedure declarations would be followed by:

```
integer stack_pointer;
integer array stack_store[1:n];
stack_pointer := 1
```

and the procedure bodies would be filled in in the obvious way.

On the other hand, if the stacks are implemented as linear lists the parameter n would be ignored and the procedure declarations would be followed by:

```
class stack_node(element); value element; integer element;

begin
    ref(stack_node) link

end;

ref(stack_node) head
```

and the procedure bodies are filled in in the appropriate way. Here is an example of the declaration of push:

```
procedure push(x); value x; integer x;

begin
    ref(stack_node) temp;
    temp :- new stack_node(x);
    temp.link :- head;
    head :- link
end of push;
```

This second declaration of stack violates restriction (2) (as enforced by the DEC-10 implementation). In spite of the fact that the accessible attributes of these two classes are the same (except, of course, in the second version the procedure full always returns true) the two class declarations are not interchangable!

The array implementation permits the use of dot notation to refer to the four accessible attributes and the second prohibits the use of dot notation. This means that the two implementations are not interchangable inspite of the fact that they are functionally equivalent. This is a consequence of restriction (2).

These examples illustrate my claim that restriction (2) limits the use of classes for program abstraction. In my opinion, this evidence supports the removal of restriction (2).

REFERENCE

[1] O.-J. Dahl, B. Myhrhang and K. Nygaard. Common Base Language. Publication No. S-22, Norwegian Computing Center, October 1970.

UNCLASSIFIED Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) (19) AFOSR-TR-80-0282 — 2. GOVT ACCESSION NO. AD-A083 266	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) CONCERNING CLASSES WITHIN CLASSES.	5. TYPE OF REPORT & PERIOD COVERED Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Richard J./Orgass	8. CONTRACT OR GRANT NUMBER(s) (15) AFOSR-79-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS Virginia Polytechnic Institute and State Univ. Department of Computer Science Washington, DC 20041 (12) 9	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) (17) 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332	12. REPORT DATE (11) 15 Jan[redacted]79
	13. NUMBER OF PAGES Eight
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (14) VPI-TM-79-1	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(9) Technical memo

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

SIMULA, program abstraction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two examples are used to show how the SIMULA restriction on the use of dot notation to reference attributes of classes limits the abstraction mechanism provided by classes.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 411 018

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)